CIS*4500 Term Project

SENTIMENT ANALYSIS

Manav Patel

0888536
2019-11-28

Introduction

Problem Identification:

The problem being tackled in this report is the domain of sentiment analysis on movie reviews. Sentiment analysis is the process of using statistical natural language processing, text analysis and linguistics to identify key traits within a user's opinion and grasp their overall sentiment towards a product or item. The problem being faced is finding the best method of extracting these features and classifying the reviews as either positive or negative. Many large companies apply sentiment analysis to quickly filter reviews about a product or service, this sort of classification can be very helpful for the consumer and the company. Extending the applications of sentiment analysis outside the domain of movie reviews, it is possible to filter opinions about an item for an ecommerce website, or gather emotional data about a population to gauge wellbeing.

The major techniques being used in this project is Data Analytics, Feature Selection, and Machine Learning Classifiers. All the classification has been implemented using Java and we Weka machine learning package. Before classifying the data, a preliminary summary of the dataset is preformed. The data analytics of the dataset provides some useful insights onto the datasets that we're dealing with. After data analytics comes feature selection. The process of feature selection is selecting a subset of terms that appear in the dataset and only selecting those terms that offer the greatest value in text classification. The two feature selection methods used are Part of Speech tagging, and Part of Speech tagging with stop word removal and document frequency cut-off. The first approach is naively selecting the most frequent adverbs and adjectives from the dataset and filtering the corpus for any words that do not belong in this set of features. The second approach is implementing a list of commonly used stop words and removing these words from the features selected. We further remove the top 10 most frequent words as we follow Zipfs law, which states that the most common words are likey to be stop words that do not provide any value for classification. Once feature selection was implemented we applied 2 text based Machine Learning models, Naïve Bayes Multinomial, and Stochastic Gradient Decent, more on the explanation about these classifiers below.

The major results outputted from the combination of feature selection and classifying methods is that for this particular dataset the Naïve Bayes Multinomial (NBM) out performed Stochastic Gradient Decent (SGD) on both feature selection methods. On average the NBM model performed at 82% F-Measure on pure PoS tagging and 80% F-Measure on PoS tagging stop word removal and document frequency cut-off. SGD performed lower on all feature selection methods than NBM, performing at an average of 78% F-Measure for pure PoS tagging and 77.5% F-Measure for PoS tagging stop word removal and document frequency cut toff.

Description of Techniques

Feature Selection

In this report we used 2 feature selection techniques Part of speech tagging and part of speech tagging with stop word removal and document frequency cut off. Part of speech tagging is the process of adding tags to each word in the dataset. To achieve this we used the open-nlp PoS tagging java library. The basic idea of PoS tagging is going through the corpus and labelling every word with a specific tag, for this domain we are only interested in adjectives and adverbs as these words dictate the sentiment of a review well. Stop word removal is the process of removing commonly used words such as “them”, “that”, “do”, etc. from the corpus hence allowing for more accurate features. Document frequency cut off is a method that is derived from Zipf’s law. Zipf’s law states that given the frequency, f, of a word and its rank, r, in the corpus then $f \propto 1/r$. Basically stating that the frequency of a term decreased very rapidly within a given corpus. With this intuition we can remove the most commonly used words in the English language and focus on only the words hat matter. In this report we remove the top 20 words that are the most frequent and evaluate that feature selection method on different classifiers.

Machine learning Classifiers

We trained 2 machine learning models that were a part of the Weka package, Naïve Bayes Multinomial (NBM) , and Stochastic Gradient Decent (SGD). The Naïve Bayes Multinomial classifier is based on Bayes theorem, which assumes a strong independence between features/terms and is stated as the following equation:

$$p(w_t|c_j) = \frac{1 + \sum_{i=1}^{|D|} n_{it} f(c_j, d_i)}{|V| + \sum_{s=1}^{|V|} \sum_{i=1}^{|D|} n_{is} f(c_j, d_i)}$$

Where d is a document in the dataset and c is a class. This approach is popular for text categorization and one of the best performing for this analysis. The second approach used is stochastic gradient decent, which is an optimization based algorithm used to find the best values to the parameters of a function that minimizes the cost. The intuition behind gradient decent is sort of like a bowl, each data point is mapped to random locations which is the cost of the coefficients when starting the algorithm. The optimization function is trying to get the data points to the center of the bowl which indicates the best coefficients which relate to the lowest cost. The stochastic version of the gradient decent function follows a random walk pattern and updates the coefficients on every training instance. The benefit of stochastic gradient decent over the normal gradient decent is speed, stochastic gradient decent can learn much faster on large datasets. The loss function being used for this implementation is the Hinge loss or (SVM) function.

Implementation Details

Feature Selection, Filtering & Splitting of Corpus:

The feature selection methods are implemented in the FeatureSeletion class. This class makes use of the PoS tagging model provided by open-nlp and essentially tags each term in the dataset and ranks them based on frequency. The class will then filter out the appropriate amount of features (500, 1000, 2000 etc.). In more detail the major data structures used in this class are two ArrayLists which correspond to the features and it's relevant tag and a TreeMap which holds the words as keys and their frequency as values. The feature selection algorithm will first load the dataset located in txt_sentoken into memory and load each word and tag encountered into two separate ArrayLists. Once every word is tagged the TreeMap will be built based on the two ArrayLists. Iterating through the list's if any word already exists in the TreeMap then that words counter will get incremented and if it does not a new entry with the word and counter set to 0 will be added to the TreeMap. Once the TreeMap is fully loaded, the program will output a features.txt file, the size of which is indicated in the command line arguments used to run the FeatureSelection class.

Weka requires files to be in .arff format and in order to get our corpus in that format we used the TextDirectoryLoader class that weka provides to convert text data into .arff format. This functionality is wrapped inside the CorpusFilter.java file which essentially will load the features.txt file outputted from the FeatureSelection class and apply that filter to the corpus and output a new directory named filtered_sentokens. In more detail the CorpusFilter class takes in one command line argument that corresponds to the feature.txt file and will load the features into a HashSet (this is mainly for efficiency for searching for feature words). Once the HashSet is populated with feature terms each document from the txt_sentoken folder will be loaded into memory and any words that are not in the HashSet will be filtered out. Once this completes the filtered dataset will be outputted to a new folder named ./filtered_sentoken. Finally the program will output a long arff file that corresponds to the entire corpus. This will be used by the SplitDataset class.

Once all the preprocessing has completed we an split the dataset into training, validation and test, the SplitDataset class will take care of this requirement. Weka comes with a Resample object which allows manipulations of arff files and data within them. A Resample object takes in a percentage which is the amount of data to pick out of a dataset. The Resample class is called on the corpus.arff file (the whole dataset) with a percentage of 85%. This will take the corpus.arff file and return a new copy named intermediate_training.arff which contains only 85% of the data as the original corpus.arff file. The program then takes the difference of between the intermediate_training.arff and the corpus.arff to obtain 15% distinct documents which represent the validation set. The same procedure is called but this time it’s for 80% which will resample the intermediate_trainin.arff into training.arff and the difference between the intermediate_training.arff and training.arff will be outputted into a test.arff file(representing the test set).

Classifier Objects

The Naïve Bayes Multinomial classifier resides within the BayesClassifier.java class and the Stochastic Gradient Decent resides within the GradientDecent.java class. These two classifiers work in similar ways and depend on the Weka packages. The BayesClassifer will load the training.arff, test.arff and validation.arff files into memory via the ArffReader class. Then will build the classifier which essentially runs the NBM classifier on the training set. Once the classifier has finished building and running on the training set it will be evaluated using 10 fold cross validation on the test set and an F-Measure will be outputted. Once the 10 fold cross validation is complete the model will be evaluated against the validation set and a final F-Measure will be outputted. The Stochastic Gradient Decent model works in very similar way but just makes use of the SGDText classifier from weka.

Results and Analysis

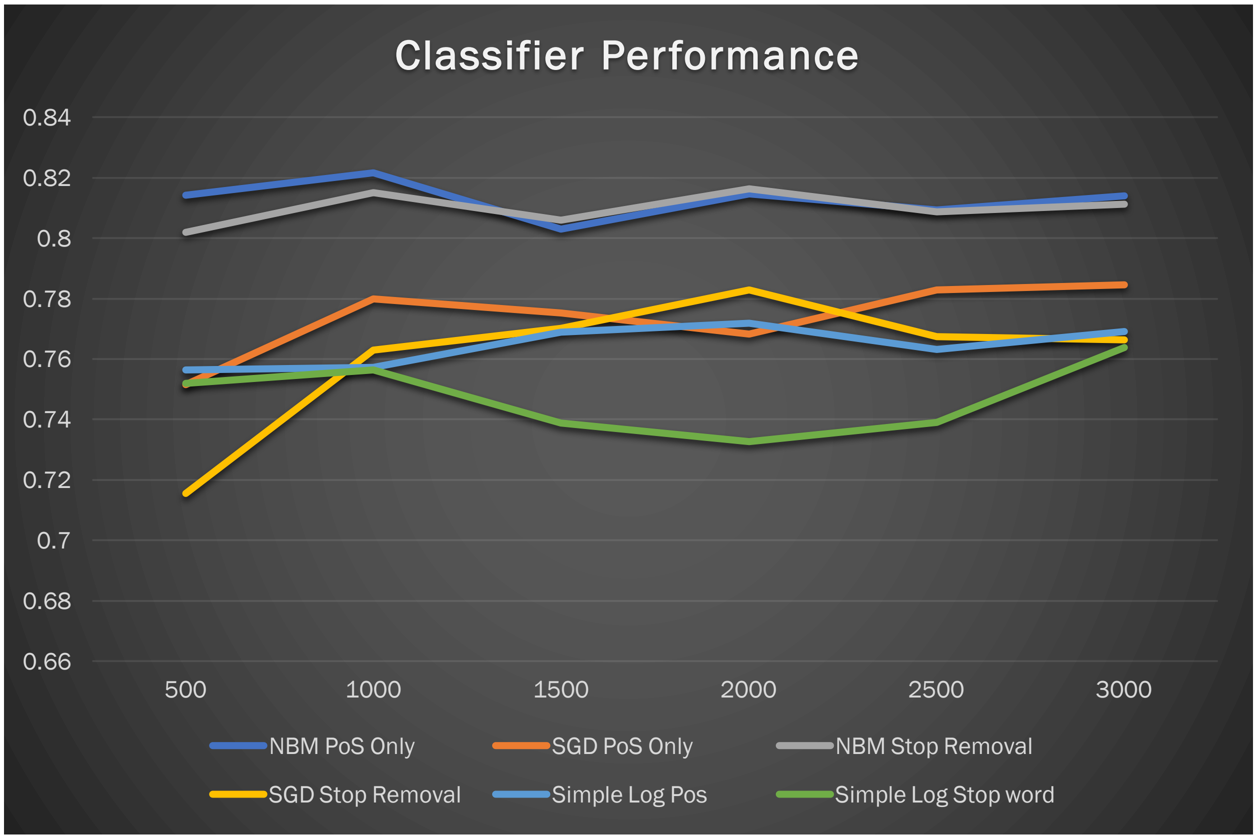

Figure 1.

Overview:

Overall the two methods of classification were fairly accurate but the most accurate being NBM (Naïve Bayes Multinomial) over all the feature selection methods. The average F-Measure for NBM for PoS tagging only is 81.28% and the average for PoS tagging with Document Frequency Cut-off is 80.98%. These two methods beat Stochastic Gradient Decent by a fair margin. The average for SGD on PoS tagging only is 77.37% and the average for SGD with stop word removal and document frequency cut off is 76.09%.

Naïve bayes with PoS tagging only would choose the most frequent adjectives and adverbs from the corpus and build a model based on these features. Since NBM is a great independent classifier for text categorization this would quite clearly be an optimal solution for this dataset. Since we're trying to categorize movie reviews as either good or bad (binary classifier) it's quite evident that the NBM is out performing the SGD model. The best performing feature size for this classifier is 1000 features which has a F-Measure of 82.16%. The worst performance feature size is 1500 which has a F-Measure of 80.29%. See Table 1 for the results

Naïve Bayes Multinomial with Stop word Removal and Document Frequency Cut off proved to be a slight under performer than using only PoS tagging. This might be because the reviews use a lot of the same adjectives to describe a movie, and cutting off the top 20 would cause a loss of important data. These best performing feature size for the NBM with Document Frequency Cut off is 2000 with and F-measure of 81.64% although this is only marginally better than a feature size of 1000 at F-Measure 81.51%.

For NBM on both feature selection methods the pattern is very interesting that the F-Measure actually decreases for a feature size of 1500 but is higher for 1000 and 2000. I would expect a trend that is generally linear. The analysis here is that there may be some overfitting when moving into 1500 feature size range but a feature size of 2000 might add some more relevant terms which would bring the model accuracy back to normal. Observe the dip in *figure 1* for NBM PoS Only line and the NBM Stop Removal line. The F-Measure is higher for 1000 and 2000 but lower for 1500.

Table 1: Naïve Bayes With PoS Tagging F-Measures

Feature Size	F-Measure
500	0.814332247557003
1000	0.821603927986906
1500	0.802980132450331
2000	0.814691151919866
2500	0.809286898839137
3000	0.813953488372093

Table 2: Naïve Bayes with Stop Word Removal and Doc Freq Cut Off

Feature Size	F-Measure
500	0.801916932907348
1000	0.814995925020374
1500	0.805921052631579
2000	0.81639344262295
2500	0.808652246256239
3000	0.81135225375626

Stochastic Gradient Decent has proven to be a under performer on all aspects for sentiment analysis. The average F-Measure for SGD with only PoS tagging is 77.37% and the average F-Measure for SGD with Document Frequency Cut off is 76.09%.

SGD with only PoS tagging proved to be the better performer for feature selection with the best feature size being 3000 at a F-Measure of 78.46% and the general trend here is that the greater the feature size the better SGD performs this is evident from figure 1. The worst Feature size for SGD is 500 with a F-Measure of 75.16%. An intuition to this may be that the Hinge function used by SGD may not have enough data points to optimize the data points, and incorporating more data points would allow for a better more accurate classifier.

SGD with PoS tagging, Stop Word removal and Document Frequency Cut Off turned out to be the worst performing classifer and feature selection method for this analysis. The average F-measure was 76.09% and the worst feature size was 500 with an F-measure of 71.56%. The best performing feature size of 2000 with an F-Measure of 78.28 performs at the average of the previous SGD with only PoS tagging. Summary of the results are listed below.

Table 3 Stochastic Gradient Decent with PoS Tagging

Feature Size	F-Measure
500	0. 751623376623376
1000	0.779853777416734
1500	0.775244299674267
2000	0.768352365415987
2500	0.782889426957223
3000	0.784602784602784

Table 4 Stochastic Gradient Decent With Post Tagging, Stop Word Removal and Doc Cut Off

Feature Size	F-Measure
500	0.71558120362737
1000	0.762920426579163
1500	0.770226537216828
2000	0.782820097244732
2500	0.767479674796748
3000	0.766447368421052

Conclusion

In conclusion the machine learning classifiers used in this assignment are very accurate in predicting sentiment of a movie review dataset. The best performing classifier is by far the Naïve Bayes Multinomial and the worst is Stochastic Gradient Decent. The best feature selection methods would have to be the PoS tagging only. In the future I would try to implement a Chi Square feature selection method and see how that works out for the classifiers and maybe implement a Multilayer Perceptron Classifier.

Build Instructions

The project is built via make and contains a lot of rules to build specific aspects of the classifier system.

Building and running Naïve Bayes Multinomial:

- Before doing anything run make.
- If you want to run the Bayes Multinomial Classifier with PoS tagging only (the first feature selection method) then run: **make run-all-bayes-500**
  - This will run the feature selection of size 500 on the Bayes Classifier and will print out the summary of the classifer and the Average F-Measure
  - If you want to run the same feature selection with a bigger feature selection size run: make run-all-bayes-2000
- If you want to run the Bayes Multinomial on the second feature selection method run: **run-all-bayes-stopword-500**
  - This will run feature selection method 2 with a feature size of 500
  - If you wish to run a greate feature size run: **run-all-bayes-stopword-2000**
- **You can see the pattern run-all-bayes-FEATURESIZE or run-all-bayes-stopword-FEATURESIZE**

Building and running Stochastic Gradient Decent

- Before doing anything run make.
- If you want to run the SGD with PoS tagging only (the first feature selection method) then run: **make run-all-sgd-500**

    - This will run the feature selection of size 500 on the Stochastic Gradient Decent and will print out the summary of the classifier and the Average F-Measure
    - If you want to run the same feature selection with a bigger feature selection size run: **make run-all-sgd-2000**
- If you want to run the SGD on the second feature selection method run: **run-all-sgd-stopword-500**
    - This will run feature selection method 2 with a feature size of 500
    - If you wish to run a greater feature size run: **make run-all-sgd-stopword-2000**
- **You can see the pattern run-all-sgd-FEATURESIZE or run-all-sgd-FEATURESIZE**

Building and running Simple Logistic Classifier

- Before doing anything run make.
- If you want to run the simple logistic classifier with PoS tagging only (the first feature selection method) then run: **make run-all-simplelog-500**
    - This will run the feature selection of size 500 on the simple logistic classifier and will print out the summary of the classifier and the Average F-Measure
    - If you want to run the same feature selection with a bigger feature selection size run: **make run-all-simplelog-2000**
- If you want to run the simple logistic on the second feature selection method run: **run-all-simplelog-stopword-500**
    - This will run feature selection method 2 with a feature size of 500
    - If you wish to run a greater feature size run: **make run-all—simplelog-stopword-2000**
- **You can see the pattern run-all-simplelog-FEATURESIZE or run-all-simplelog-stopword-FEATURESIZE**

How to generate the list of features based on pos tagging

- make run-feature
    - By default this will output a feature selection list of 1000

How to generate the list of test.arff validation.arff and test.arff

- make run-corpusfilter
- then make run-splitdataset
    - this will filter the data that lives in ./text_sentoken with the feature file from the command above and output it to filtered_sentoken/

Bibliography

Brownlee, J. (2019, 11 28). *Machine Learning Mastery*. Retrieved from https://machinelearningmastery.com: https://machinelearningmastery.com

Stanford. (2019). *nlp.stanford.edu*. Retrieved from Stanford NLP: https://nlp.stanford.edu/IR-book/html/htmledition/contents-1.html

Synced. (2019). *Applying Multinomial Naive Bayes to NLP Problems: A Practical Explanation.* Toronto, Ontario, Canada: Medium.com.